System Testing

By K. A. HELLER, D. A. SCHMITT and R. M. TAYLOR

(Manuscript received August 17, 1970)

The results of system testing against a set of performance criteria indicate the readiness of a new installation for service. In the first installation of TSPS No. 1 at Morristown, N. J., system testing also had to accomplish a design shakedown concurrent with final design and program debugging efforts at the Holmdel laboratory test model. This article describes the overall efforts and test facilities involved in the initial shakedown.

I. INTRONUCTION

In a typical TSPS No. 1 installation, the Western Electric Company performs a series of installation tests which check out the equipment as received from the factory and the equipment interconnection cabling as installed in the field. Extensive use is made of manual buzz through testing and X-ray programs.[1] Then system tests are conducted to demonstrate that the system is completc and operable. Initially basic system functions related to handling a small call load are sbaken down. Then all system features are tested in tbe presence of the call load background. Finally, continuous operation is imposed wherein the system is allowed to run under typical environmental conditions with call load boxes being used to simulate traffic. System performance during continuous operation must satisfy a set of performance criteria (described in Section III) to be acceptable for turnover to the Bell System Operating Company. After turnover, operating company craftsmen tie tbe new system into the existing Bell System plant, determine that connecting offices function smoothly with it, run it continuously in accordance with telephone company discipline, and for training purposes perform all routine functions. On completion, the system is cut over into service.

The system tests applied at the first TSPS No. 1 installation at Morristown, N. J., were more exhaustive than would be used for sub-

sequent installations since, in addition to demonstrating a complete and operable installation, they also had to demonstrate an adequate design. Also the performance criteria for turnover to be used in subsequent installations had to be established on the basis of experience gained with the system at Morristown. As a result, considerable engineering judgment had to be exercised in deciding when to cut over the first installation. The adequacy of the system's performance to give a commercial grade of service in spite of residual design and installation bugs had to be balanced against economic and service demand pressures for an early cutover.

System testing in the first installation turns out to be an engineering feat. The planned tests reveal design bugs which necessitate software and/or hardware changes. In addition, the effectiveness of the tests devised for the newly designed system is not always predictable, with the result that the tests themselves have to be revised. The time and activity required to design and install corrective changes interferes with further testing. Almost constant re-evaluation of test plans, schedules, and status are necessary.

In the year prior to the Morristown TSPS No. 1 cutover, a total of 177 necessary hardware changes were installed. Most of these were minor, involving only a few wire changes. Some, however, were major, such as complete replacement of store frames, bus connectorization, and several significant changes in the position subsystem equipment. In the same year over 2000 trouble reports were issued that resulted in software changes.

Software debugging and utility facilities (see Section 2.2.1) were left in the Morristown system until three months before cutover. In addition to providing debugging aids, these facilitated timely insertion of program changes. After removal of the utility programs, special generic recent change programs were used to implement program changes. These had been provided to facilitate emergency changes in working offices and proved useful for pre-cutover changes at Morristown. As program overwrites or changes accumulated in the program memory, it became necessary to reassemble the programs to permanently incorporate the changes and produce new issues of program load tapes and listings. New program issues were loaded in the test model at the Holmdel, N. J., location of Bell Laboratories, and after initial shakedown were loaded at Morristown. Program reload and recovery of the system to the operational state that existed prior to the reload frequently took several days, particularly in the early stages of system testing.

1.1 *Program Administration*

Because of the volume of program change activity, a program change committee was formed about a year before the Morristown cutover. As programs became debugged, they were "frozen," and thereafter changes could be made only with the change committee's approval. To insure tight control, the program "source" card decks for all frozen programs were turned over to a program support group which had the responsibility thereafter for reassembling frozen programs to incorporate only approved changes and for producing new program issues.

When the need for a program change became known, the programmer would originate a correction or program overwrite, which he would prepare in the form of a punched card deck. By means of the utility system he would insert the overwrite into the system's memory on a temporary basis for testing. If good, he would submit the overwrite deck, a correction report, and an alter deck to the change committee for approval. The alter deck was required for incorporating the change in the next program reassembly. If approved, the overwrite would be inserted into the system memory via the utility system on a permanent basis. When a sufficient number of overwrites had accumulated, the new program issue was produced and loaded.

To minimize programmer effort and human error, the utility and program assembly systems were designed so that overwrite and alter decks could be identical in content and in addition could be in symbolic format except for certain items that must be in absolute code format for the overwrite deck.

Another useful programmer aid that was built into the assembler/loader system is the ability to generate program listings that are consistent with the state of the system's memory as modified by overwrites. Specifically, the instructions are listed as corrected by the overwrites and in the sequence executed, but the address that appears for each instruction is the absolute location of the instruction in memory. This listing is particularly useful to the programmer while overwritten programs reside in the system memory. Later, when the programs are reassembled for a new issue, the instructions are resequenced so that successive instructions occupy successive memory locations.

II. PROGRAM DEBUGGING

Debugging of the Stored Program Control-TSPS software package

occurred in three stages, each of which required a different set of debugging tools and techniques:

Stage 1—Strip or T-cart debugging
Stage 2—Utility debugging
Stage 3—Functional testing

Figure 1 shows when these three stages were employed and the fact that they overlapped in time. At most times during the entire debugging interval, the tools and techniques of all three stages were in use. The transition from one stage to the next was not prearranged but rather represented a natural evolution. Actually the three stages were followed for each program. Tbe fact that programs became available for debugging at various times and underwent extensive changes at times is the primary cause of the extensive overlap.

2.1 *Strip or T-Cart Debugging*

Strip debugging is the process of checking basic "strips" of program instructions without testing the interconnections between the strips. Such a test is normally conducted by setting up the initial conditions required by the strip being tested, executing the strip, and then observing the terminal conditions.

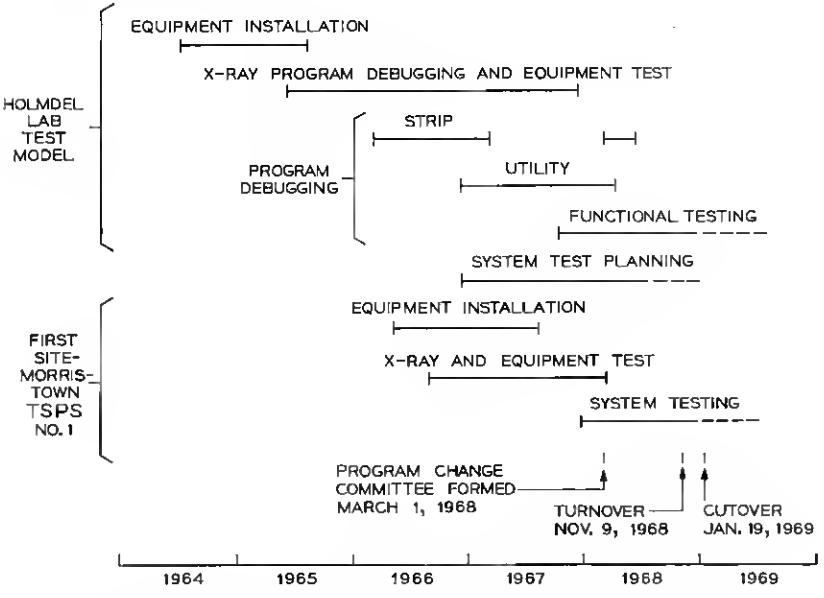

Fig. 1—Test model and first site installation and test schedule.

Strip debugging was used to check the basic SPC-TSPS control programs and the utility package which was later used for utility and batch-mode debugging operations and functional testing. The only debugging tool required for strip debugging is the test console (T-cart) shown in Fig. 2. All initial conditions are established through the T-cart instruction and data insertion keys. The instructions being tested are then executed one at a time by successively operating the cycle key. The results are observed on the various display lamps. It is also possible to execute the instructions at normal machine speed by operating the run key. For this case, the T-cart is equipped with four hardware matchers which can be set up to stop the machine when:

(*i*) a specified address is accessed for reading, for writing, or for program execution;

(*ii*) an address within or without a specified range is accessed for reading, for writing, or for program execution; or

(*iii*) 20 bits of data being written or read from temporary memory match pre-specified data.

When the machine is stopped by one of the matchers, the results of the program operation can be examined by reviewing the T-cart display lamps.

Strip debugging enjoyed heavy use only during the very early part of the debugging interval. As more programmers desired access to the laboratory model, the obvious inefficiencies of the manual strip method began to seriously retard progress. However, at later times during the debugging interval programmers sometimes found it necessary to fall back on the strip method. This was particularly true during the testing of the processor and store fault recognition programs[2] which were not as amenable to utility debugging.

2.2 *Utility Debugging*

In the utility debugging mode, the laboratory model is under control of the "utility program package." This is a collection of programs that coordinate the execution of each debugging job. All commands to the utility package are entered through a card reader which is connected to the laboratory model through a special interface. All output is directed to a high-speed printer, which is also specially interfaced to the laboratory model.

To use the utility debugging system, the programmer prepares "debugging jobs," in the form of punched-card decks. The card decks

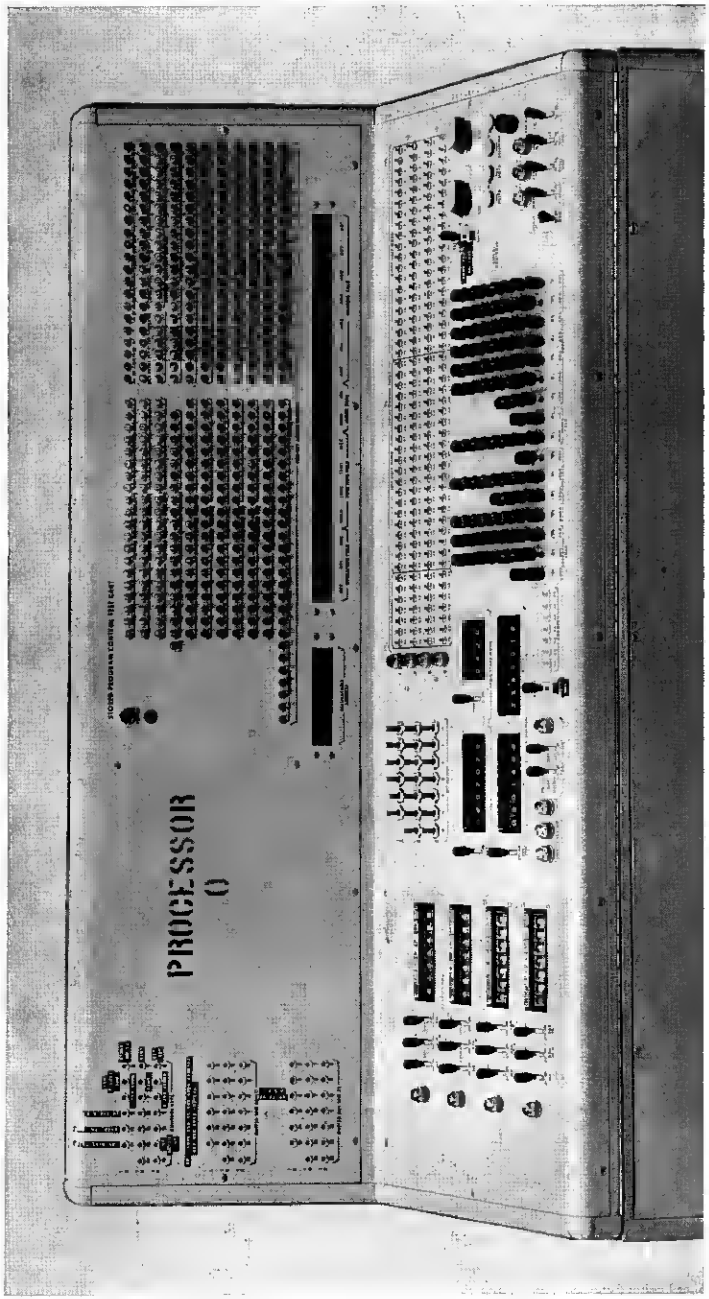

Fig. 2—Test cart.

contain control or command cards which are read and executed by the "utility program." These commands establish initial conditions, generate printouts of specific registers and memory areas, and control the execution of the instruction strip being tested. In other words, all of the manual functions required for manual strip debugging are prespecified on punched cards and executed automatically at high speed.

Of prime importance to the utility mode are the matchers which trigger execution of the various commands as prespecified in the punched card decks. The T-cart matchers can be used for this purpose. To increase the number of available matchers and to avoid the manual setup time for the T-cart matchers, twelve pseudo matchers were provided as part of the utility system. The pseudo matchers permit the programmer to specify the program address to match in his debugging deck. The utility program changes the instruction at that address to one which causes the processor to interrupt the normal sequence of program instructions at A-level or the highest level interrupt.[2] Whenever the program under test reaches that address an interrupt occurs, control is transferred to the utility program, and the required utility functions are executed. Then the original instruction which had been saved by the utility program is executed and control is transferred back to the system program.

Because the utility system enabled the programmers to completely plan and specify their debugging jobs on punched cards, a batch mode type of debugging operation was made possible.[3] In the batch mode the programmers submitted their relatively short debugging jobs to the laboratory support group in lieu of requesting machine time to run their own jobs. The support group batched the submitted jobs and ran them in rapid sequence, returning the results to the programmers for analysis. This technique makes more efficient use of machine time than if each individual programmer conducted his own debugging runs and further permits a programmer to get several runs a day, which might not otherwise be possible.

The batch mode technique was first tried on the No. 1 ESS ADF (Arranged with Data Features) test model in New York City. Its success there prompted its use in TSPS No. 1 where it also proved to be successful.

For the more complex debugging runs requiring more time per run or programmer presence at run time, the programmers would sign up for machine time, which would be scheduled by the laboratory support group. Even for those cases where programmers signed up for

machine time they still prepared their debugging runs and punched their utility control cards in advance to take full advantage of the utility system and maximize machine time utilization.

In the final analysis, maximum debugging efficiency (orders debugged per unit time) is achieved only if the programmer uses the machine to get the results of pre-planned debugging runs and does his analyzation of results at his desk. Machine time is wasted when a programmer signs up for a large block of time and then spends a large part of the time analyzing results to plan the next step. The success of the batch mode operation is largely attributed to the fact that it overcomes loss of machine time due to analyzation activity by the programmer while he is assigned exclusive use of the machine.

2.2.1 *Utility Package*

This section gives a detailed description of the functions provided by the utility program package. The utility programs are loaded into spare SPC memory areas and, where necessary, temporary linkages are provided to the generic program software.

The utility package is divided into four main parts: (a) initialization, (b) command processor, (c) administration, and (d) overwrite control. Throughout the system debugging interval, these programs underwent continual modification to improve their speed and power. Many improvements were suggested and implemented as a direct result of debugging experience so that the final versions bore little resemblance to the originals. The following paragraphs describe the final versions.

2.2.1.1 *Initialization Program.* The utility initialization program acts as a "system reset." That is, whenever the operator actuates the manual interrupt (MINT) key on the T-cart, the initialization program restores all hardware to service, clears all call memory, and removes all temporary overwrites from the SPC-TSPS package. At the end of initialization the system is left in the idle mode (under utility control) awaiting input from the card reader. In the batch-mode, the system is initialized between each job.

Before the generic recovery programs were debugged, the utility initialization program was also used to initialize many generic tables and software registers used by the generic programs. This feature was provided so that programmers would not have to include an excessive number of standard initialization commands in their debugging decks. As the generic program package became more reliable, these special

initialization routines were deleted from the utilities in deference to the generic initialization or restart routines.[4]

2.2.1.2 *Command Processing.* The command processing routines interpret tbe commands in the debugging deck and perform the specified functions at match time. The major functions provided are: (*i*) memory dump, (*ii*) transfer trace, (*iii*) initialization, (*iv*) jumps, and (*v*) conditional statements.

The dump function provides octal, hinary, or decimal printouts of specific memory areas when a match occurs. The addresses to be dumped can he stated explicitly in the dehugging deck, or the deck can identify the location that contains the start address for the dump, a feature known as indirect dumping. The trace function enables the programmer to obtain a printout of the "from" and "to" addresses and index register contents on each transfer instruction. Initialization functions enable the programmer to initialize index registers and memory contents to specific values before entering the test strip or to change register and memory contents during the test. The jump function is used primarily to enter the test strip initially and to reenter it (with different input data) when the end of the strip is reached. Conditional functions provide selective control over the debugging run hy allowing the programmer to specify tbe conditions under which other functions should be executed. For example, tbe programmer can specify that when matcher 6 fires the tenth time, the X register should be initialized to a value of 3 if tbe Y register is not equal to zero.

2.2.1.3 *Administration.* Part of the utility package consists of administrative routines which are used only hy the Laboratory Support Group. These routines provide for making tape copies of memory and matching or reloading these tapes. Also, the administration routines provide printouts of the size and location of patch and spare memory areas. The Laboratory Support Group required these facilities in order to maintain rigid control over the contents of memory in tbe laboratory model.

2.2.1.4 *Overwrite Control.* The overwrite control routines allow program changes to be inserted in a symbolic format. Thus the programmer can concentrate on solving his problem rather than on manually translating his symholic instructions into machine code. The overwrite program also automatically allocates all spare memory required when additional instructions are added to a program. Changes can he inserted either temporarily or permanently. Temporary overwrites are

automatically removed by the initialization program, and are always used on batch debugging jobs. Permanent overwrites are inserted only by the Laboratory Support Group.

2.3 *Functional Testing*

The normal method of debugging is to first check several basic instruction strips and then to combine these into a larger strip and check it, and so on. Eventually the strips become large enough that they constitute a set of "functional packages," such as the call connections program. At this point utility debugging diminishes and the functional test period begins.

A "functional" test differs from a "strip" test in that the initial and terminal conditions for the functional test cannot be handled by the utility package or the T-cart. For example, a functional test of the call connections program involves the actual seizure of a trunk and observance that the relays and network operated under control of the program. This need to have "actual," or realistic, conditions at the time of the test meant that the programmer had to be present for the test and that the test would take much longer than the few minutes normally allowed for a utility debugging run. Extensive use was still made of the utility system, however, in conducting functional tests.

Figure 3 summarizes the program debugging progress in orders debugged, using the three stages of debugging technology described. During the functional test period, the number of orders debugged per unit time was less because more sophisticated and time-consuming tests were needed to isolate the remaining problems.

III. SYSTEM TESTING

Many individual tests are performed on the various units and programs that constitute a complete system such as TSPS No. 1, but system testing properly refers only to those tests that treat the system as an entity and where the stimuli and measurements are made at the system ports. The principle results from this type of testing are the measurements made to ensure that an acceptable system performance objective is met.

Planning for this testing sequence began during late 1966, as shown in Fig. 1, and continued throughout the testing interval until the machine was turned over to the operating company in late 1968. During the early stages of the test planning for TSPS No. 1 it was decided to shakedown first only those features required to allow the system

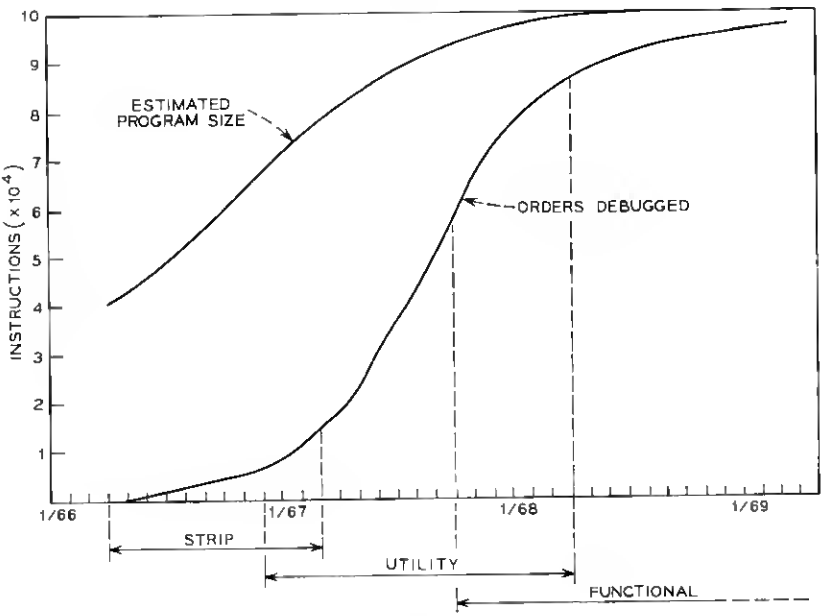

Fig. 3—Debugging progress curves.

to handle a simple call load as generated by load boxes. All subsequent testing could then take place with the system continuing to process traffic, and any interference with the systems ability to process calls would immediately point out design conflicts so that corrective action could be taken.

The call load background environment was the closest possible practical approach to the machine's normal real world environment. It was not intended that the testing done during this interval be done in the presence of an overload or even a full load condition, but rather that the system be continually exercised at a reasonable level of call load activity. As the testing progressed, the type of calls generated for the background were expanded to include as many variations as the available test equipment would allow.

In order to implement this test plan, two hardware test sets were developed:

The Automatic Local Toll Simulator Set (Fig. 4) was developed to interconnect with the TSPS incoming trunks at the main cross-connection frame terminations. This "load box" provides for the

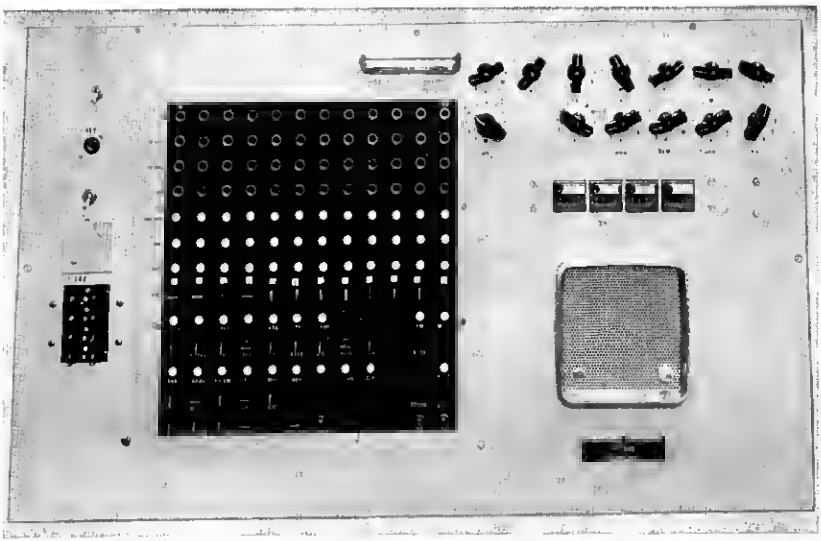

Fig. 4—Automatic local-toll simulator set.

origination and termination of calls[5] through the system. It generates any type of call that the system can process and verifies each system response to the external stimuli offered. Sufficient sets were provided for Morristown to allow the generation of approximately 4000 calls per hour of all types.

The Automatic Operator Simulator Set (Fig. 5) was developed to interconnect with the 100B Traffic Service Position. It electrically detects the various lamp displays and generates an appropriate operator's keying response, allowing the system to complete processing of the call. This set processes any type of call that can be connected to a position.

As shown in Fig. 6, these sets were connected external to the TSPS No. 1 system—i.e., they did not become a part of the system or change its operating characteristics in any way.

The test sets made possible the continual call loading of the system over the relatively long period of time during which the other operational, maintenance, and administrative tests were performed. They also provided data for one of the key system performance measurements; the Call Failure Rate.

During the test planning phase the call processing and administrative tests were written in detailed step-by-step form, and the main-

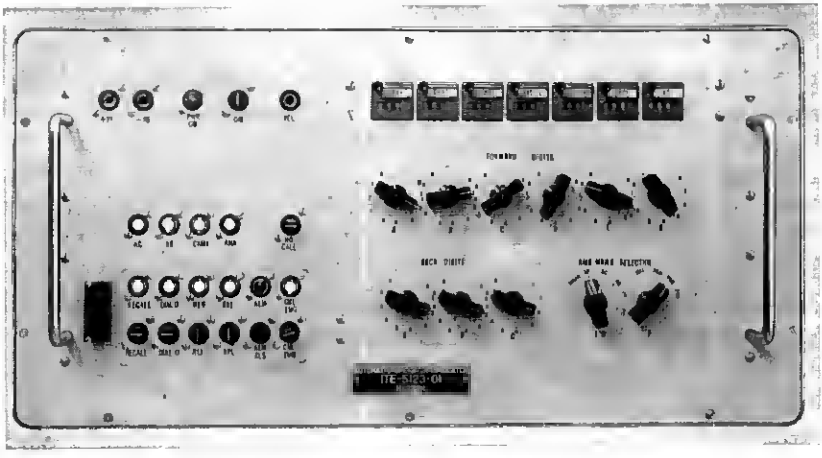

Fig. 5—Automatic operator simulator set.

tenance tests were adapted from the individual programmer's functional tests. In total, there were 345 individual tests or test items that formed the system test specification for the Morristown office and a selected sub-set has been produced in the form of Western Electric Company Installation Handbook Sections for the system testing of later offices. Since the installation at Morristown was used to prove-in the system design, the overall system testing performed was consid-

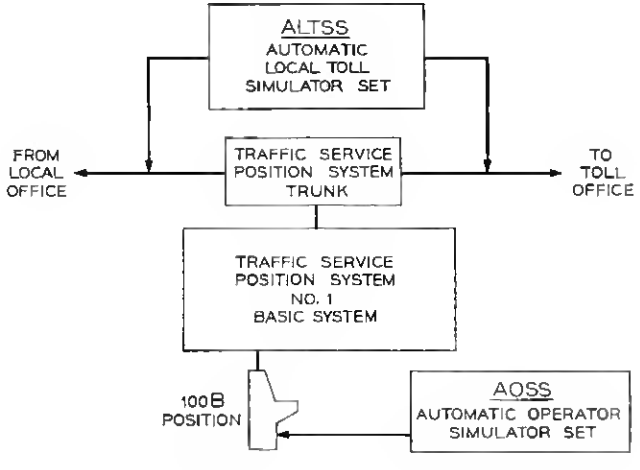

Fig. 6—Connection of ALTSS and AOSS to system.

erably more detailed and intensive than is required for later installations.

As the testing progressed, the machine was brought under call background load in late March of 1968, and this background was increased to include all call types by April. It should not be assumed that the machine was under continual call loading from this time until cutover. The call background was provided to enable the detection of program and hardware interaction problems, and as these were detected, various hardware and software techniques (utilities, etc.) were used to isolate and assist in the required solutions. Continual hardware and software changes were implemented during this interval, and entirely new programs were added to the system as they became available. When large changes were made, application of a sub-set of the detailed call processing and administrative tests, observation of the call failure rate and maintenance TTY output were instrumental in determining the goodness of the system following the change.

Throughout most of this interval the primary tool for locating and clearing software problems was the utility program package, described in Section 2.2.1. It quickly became apparent that the use of the utilities was incompatible with the call background due to the consumption of real time by the utility program. The output of the utility program is normally via the high-speed printer, and this work is done on an A-level interrupt basis. While in A-level and printing the utility results, so much time is used that the system is blinded to

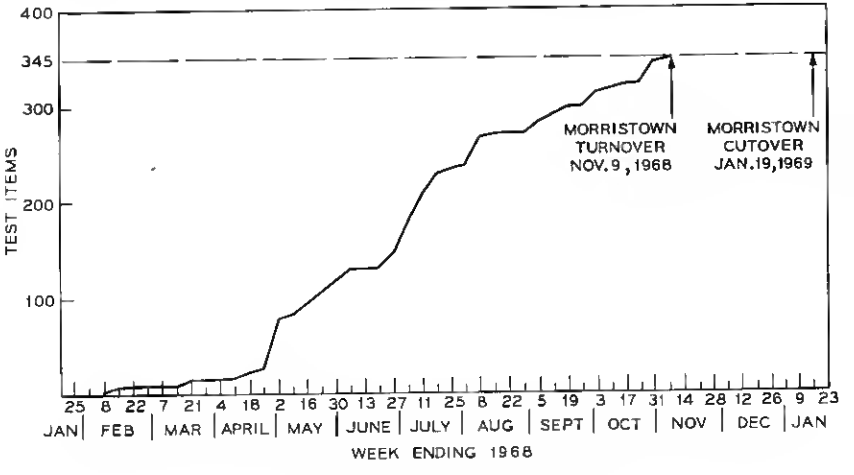

Fig. 7—System test items completed.

external stimuli and the call processing function is affected. Early in the testing interval, this interference could be tolerated, but later in many cases the call background was required to develop the problem, and because of this conflict, the data dump store and hardware transfer trace were provided in place of the utility package (see Section IV).

All during the testing interval, progress was marked by various indicators, including the Call Failure Rate, Test Items Completed (see Fig. 7), hours of simplex or duplex operation, hours of call loading, various maintenance printouts from the teletypewriter in the Maintenance Center, and the number of outstanding trouble reports. As turnover approached, the important measurements were the call failure rate, and the numbers of maintenance interrupts,[2] audit messages,[4] peripheral unit failures,[2] position subsystem failures,[2] and outstanding trouble reports. As these items approached a reasonable level, turnover to the Telephone Company took place and a new stage of testing was begun.

Following turnover, the New Jersey Bell Telephone Company performed tests on the machine, including the handling of live (non-simulated) test call traffic through the system while performing many maintenance and administrative routine operations. As was to be expected, this testing turned up problems that had not or could not have been detected previously. An intensive effort was required to clear these problems before cutover.

In spite of a few known residual design problems, the Morristown TSPS No. 1 system was cut over into service on January 19, 1969, and

TABLE I—TSPS No. 1—PERFORMANCE CRITERIA MEASURED OVER A 48-HOUR PERIOD

	Requirements for Morristown	Requirements after 5/1/70
Call Non-Completion Rate*	≦ 0.5%	0.3%
Audit Errors	≦ 250/day	50
Maintenance Interrupts		
Software	≦ 3/day	1
Hardware	≦ 10/day	4
System Recovery Phases		
MNA, MJA	≦ 2/day	0
SIA, SIB	≦ 0	0
Store Errors	≦ 2/day	2
SPC Faults	≦ 10/day	5
Peripheral Unit Faults	≦ 20/day	10
Position Subsystem Faults	≦ 10/day/group	10

* Includes an allowance for test equipment irregularities.

has provided a satisfactory grade of service ever since. As expected, customer usage of the system revealed design problems that had not been detected by earlier testing. A concerted effort had to be applied in the first few weeks after cutover to clear these problems.

Based upon Morristown system experience, the performance criteria were established for all new systems. These are summarized in Table I. Basically the performance criteria establish an upper limit for the number of unexplainable trouble or fault indicating events. If a particular threshold is exceeded but the excess can be attributed to a source external to the TSPS No. 1 system or can be attributed to an explainable procedural error, the system performance is considered acceptable.

IV. FIELD SUPPORT AND HARDWARE MONITORING FACILITIES

As one might expect, no reasonable amount of design shakedown or system testing can guarantee a trouble-free design for a system as complex as TSPS No. 1. Tbe combinations of key actions which the TSPS operator can generate individually and in concert with the other operators served by the system are practically unlimited. As a result, a field support operation that continues until design problems no longer interfere with field operation must be provided. To effectively support the field, tools for obtaining the right kind of data to allow design trouble analysis must be provided, and the tools must not interfere with basic system operation.

4.1 *Dump Store and Transfer Trace*

The two most vital sets of data required in a stored program control system to isolate design problems are the state of the system's memory at the time of the interesting event—the time the problem or trouble manifests itself—and the program sequence leading to the interesting event. To satisfy this need, the data dump store* and hardware transfer trace were developed. The dump store is an autonomous facility that continuously monitors tbe state of the call or transient

* The data dump store concept originated with the No. 1 ESS development. For TSPS No. 1 a specially modified PBT store or store pair is connected to one of the store buses to monitor all processor writes to the unprotected memory in the system stores. Since the unprotected memory area is assigned to various 16ths of the systems stores,[6] each 16th of the dump store memory is equipped with special switches that permit specifying the system store name code and 16th that is to be monitored by each 16th of the dump store memory. Also, in a write-only mode the dump store can be made to accept writes at a rate faster than a system store in a normal read/write mode. As a result, the dump store can accept all store writes at the system write rate.

data stored in memory. The transfer trace continuously records in its autonomous memory the FROM and TO addresses of the last 80 transfer instructions executed hy tbe program system. At the time of an interesting event, both the dump store and transfer trace are "frozen" with their present contents. Later the stored information is extracted for offline analysis and both devices are reset to record data for the next interesting event. An offline program was developed to organize and format the data to ease tbe joh of interpretation and manual analysis.

The interesting event is detected by setting program address matchers to fire, thereby freezing the dump store and transfer trace. As a result the state of the system with the trouble condition present is preserved in the monitoring devices without in any way interfering with system operation or recovery.

Information gathered from the dump store and transfer trace has been instrumental in solving most post cutover design problems in the TSPS No. 1 system. In the first 20 weeks following the Morristown cutover ahout 100 dump store tapes and transfer traces were processed. In most cases they contained sufficient information to permit complete prohlem solution or they narrowed the scope of the prohlem considerably. In the latter case, the program address matchers were reset to fire at program locations which could provide more data in the narrowed problem area. This is the significant advantage of using program address matchers as interesting event detectors.

Because the original 80 transfer capacity transfer trace was so effective, an improved version has been developed wbicb provides for storing the last 600 FROM/TO transfer addresses or the last 100 transfer addresses plus the contents of the processor index registers at the time of each transfer.

V. EPILOGUE

System testing requires a thorough understanding of all the functional and maintenance requirements of the system and requires early and careful planning. The people who are going to he tbe system testers should be identified early to allow them to broaden their knowledge to be effective as system testers. Test plans should be made early enough to insure that test facilities like the load boxes and dump stores are available when needed. System testers sbould have a reasonahle understanding of existing systems of time-proven design tbat are going to interconnect with the newly designed system. Tbere is

always the tendency to place fault on the new system when interconnection problems develop. Experience has shown that the interconnecting system is often at fault and the troubles are revealed because the new system has more sophisticated maintenance features or is less tolerant of equipment that only marginally complies with specified operating criteria. Everyone concerned with system testing must maintain an unbiased position.

Finally it should be recognized that exhaustive system testing for design shakedown does not end with the first installation. Whenever a new system feature is added or system capability is exploited for the first time in the field, design shakedown testing must be performed. For example, the second TSPS No. 1 site at Miami was the first to employ multiple remote position subsystems and the third site at Houston was the first that is large enough to permit testing to determine if the system has sufficient real-time capacity to handle design limit traffic loads. Also shakedown testing must be performed to debug growth procedures for adding equipment to a working office whenever equipment of a particular type is added for the first time. Design shakedown testing ends only after all system features and capabilities have been fully exploited in the field and the system is capable of smooth field operation requiring a minimum of manual intervention and maintenance. This is achieved only through continued diligent field support.

REFERENCES

1. Haugh, G., Tsiang, S. H., and Zimmerman, L., "System Testing of the No. 1 Electronic Switching System," B.S.T.J., *43*, No. 5 (September 1964), pp. 2575–2592.
2. Durney, G. R., Kettler, H. W., Prell, E. M., Riddell, G., and Rohn, W. B., "TSPS No. 1: Stored Program Control No. 1A," B.S.T.J., this issue, pp. 2445–2507.
3. Barney, D. R., Giloth, P. K., and Kienzle, H. G., "No. 1 ESS ADF: System Testing and Early Field Operation Experience," B.S.T.J., this issue, pp. 2975–3004.
4. Kettley, A. W., Pasternak, E. J., and Sikorsky, M. F., "TSPS No. 1: Operational Programs," B.S.T.J., this issue, pp. 2625–2683.
5. Jaeger, R. J., Jr., and Joel, A. E., Jr., "TSPS No. 1: System Organization and Objectives," B.S.T.J., this issue, pp. 2417–2443.
6. Baker, W. A., Culp, G. A., Kinder, G. W., and Myers, F. H., "TSPS No. 1: Stored Program Control No. 1A Store," B.S.T.J., this issue, pp. 2509–2560.